[Peeking under the hood of High Availability](http://community.citrix.com/blogs/citrite/anilma/2008/09/17/Peeking%2Bunder%2Bthe%2Bhood%2Bof%2BHigh%2BAvailability)

[ Tags: [xenserver](http://community.citrix.com/blogs/tag/xenserver) ,   [ha](http://community.citrix.com/blogs/tag/ha) ]

posted in [XenServer - Server Virtualization](http://community.citrix.com/blogs/group/server-virtualization) by [Anil Madhavapeddy](http://community.citrix.com/blogs/citrite/anilma/profile)

Well, the big launch of [XenServer 5](http://www.xenserver5.com/) has gone smoothly, and with it have arrived a flood of questions about how exactly the new [High Availability](http://xenserver5.com/ha.php) functionality works.  I'll use this post to explain the overall architecture of HA in XenServer 5, and also how some of the fault detection and failure planning works.

Fundamentally, HA is about making sure important VMs are always running on a resource pool. There are two aspects to this: reliably **detecting host failure**, and computing a **failure plan** to deal with swift recovery.

Detecting host failure reliably is difficult since you need to remotely distinguish between a host disappearing for a while versus exploding in a ball of flames.  If we mistakenly decide that a master host has broken down and elect a new master in its place, there may be unpredictable results if the original host were to make a comeback!   Similarly, if there is a network issue and a resource pool splits into two equal halves, we need to ensure that only one half accesses the shared storage and not both simultaneously.

**Heartbeating for availability**

[](http://community.citrix.com/download/attachments/43614779/ha-wizard-3b.png)

We solve all these problems in XenServer by having two mechanisms: a **storage heartbeat** and a **network heartbeat**. When you enable HA in a pool, you must nominate an iSCSI or FC storage repository to be the heartbeat SR. XenServer automatically creates a couple of small virtual disks in this SR. The first disk is used by every physical host in the resource pool as a **shared quorum disk**. Each host allocates itself a unique block in the shared disk and regularly writes to the block to indicate that it is alive.

I asked [Dave 'highly available' Scott](http://dave.recoil.org/), the principal engineer behind HA about the startup process:

*"When HA starts up, all hosts exchange data over both network and storage channels, indicating which hosts \*they\* can see over both channels; i.e. which I/O paths are working and which are not.  This liveness information is exchanged until a fixed point is reached and all of the hosts are satisfied that they are in agreement about what they can see.  When this happens, the HA functionality is 'armed' and the pool is protected."*

This HA arming process can take a few minutes to settle for larger pools, but is only required when HA is first enabled.

Once HA is active, each host regularly writes storage updates to the heartbeat virtual disk, and network packets over the management interface.  It is vital to ensure that network adapters are [bonded](http://docs.xensource.com/XenServer/5.0.0/1.0/en_gb/reference.html#networking-standalone_host_config-bonds) for resilience, and that storage interfaces are using [dynamic multipathing](http://docs.xensource.com/XenServer/5.0.0/1.0/en_gb/reference.html#id2557754) where supported.  This will ensure that any single adapter or wiring failures do not result in any availability issues.

[](http://community.citrix.com/download/attachments/43614779/ha-wizard-5.png)

The worst-case scenario for HA is the situation where a host is thought to be off-line but is actually still writing to the shared storage, since this can result in corruption of persistent data.  In order to prevent this situation without requiring active power strip control, we implemented **hypervisor-level fencing**.  This is a Xen modification which will hard-power the host off at a very low-level if it doesn't hear regularly from a watchdog process running in the control domain.  Since it is implemented at a very low-level, this also covers the case where the control domain becomes unresponsive for any reason.

Hosts will self-fence (i.e. power off and restart) in the event of any heartbeat failure unless any of the following hold true:

* The storage heartbeat is present for all hosts but the network has partitioned (so that there are now two groups of hosts).  In this case, all of the hosts which are members of the largest network partition stay running, and the hosts in the smaller network partition self-fence.  The assumption here is that the network outage has isolated the VMs, and they ought to be restarted on a host with working networking.  If the network partitions are exactly the same size, then only one of them will self-fence according to a stable selection function.
* If the storage heartbeat goes away but the network heartbeat remains, then the hosts check to see if they can see all other hosts over the network.  If this condition holds true, then the hosts remain running on the assumption that the storage heartbeat server has gone away.  This doesn't compromise VM safety, but any network glitches will result in fencing since that would mean both heartbeats have disappeared.

**Planning for failure**

The heartbeat system gives us reliable notification of host failure, and so we move onto the second step of HA: capacity planning for failure.

A resource pool consists of several physical hosts (say, 16), each with potentially different amounts of host memory and a different number of running VMs.  In order to ensure that no single host failure will result in the VMs on that host being unrestartable (e.g. due to insufficient memory on any other host), the XenServer pool dynamically computes a **failure plan** which calculates the actions that would be taken on any host failure.

But there's one more complexity... a single host failure plan does not cover more advanced cases such as network partitions which take out entire groups of hosts.  It would be very useful to be able to create a plan that could tolerate more than a single host failure, so that administrators could ignore the first host failure and be safe in the knowledge that (for example) three more hosts could fail before the pool runs out of spare capacity.

That's exactly what we do in XenServer... the resource pool *dynamically* computes a failure plan which considers the "number of host failures to tolerate" (or *nhtol*).  This represents the number of disposable servers in a pool for a given set of protected VMs.

The planning algorithms are pretty complex, since doing a brute force search of all possible failures across all hosts across all VMs is an exponential problem.  We apply heuristics to ensure we can compute a plan in a reasonably small time:

* for up to 3 host failures, we do a comprehensive search which tries almost all permutations.  This covers corner cases such as having hosts or VMs with very different amounts of memory (e.g. 4GB vs 128GB).  Rather than calculate memory slots or otherwise approximate results, we just deal with them individually and give very accurate plans.
* for greater than 3 host failures, we make conservative decisions by approximating every VM to be as large as the largest, and considering each host to be the same as the most densely packed host.  We do not approximate the host memory, and so having pools with uneven amounts of host memory will be fine.  However, in approximate planning mode having a single very large VM will result in a low *nhtol* value.  If this is a problem, then try to reduce the *nhtol* or try to have a more even spread of VM memory sizes.

Since planning algorithms are designed for unexpected host failures, we only consider absolutely essential resource reservations which would prevent the VM from starting on the alternative host (e.g. storage is visible, and enough memory is present).  We do not perform CPU reservation on the basis that it can be optimised at a later stage via live relocation once the VM is back up and running.

**Overcommit protection**

We now have HA armed and a failover plan for our VMs.  But what if you want to make changes to your configuration after HA is enabled?  This is dealt with via **overcommit protection**.

The XenServer pool dynamically calculates a new failover plan in response to every XenAPI call which would affect it (e.g. starting a new VM).  If a new plan cannot be calculated due to insufficient resources across the pool, the XenServer will return an **overcommitment** error message to the client which blocks the operation.

**The "What if?" Machine**

[](http://community.citrix.com/download/attachments/43614779/ha-wizard-4b.png)

This overcommit protection would be quite irritating if you have to keep trying things and seeing if a plan exists or not, and so we built in a "[What If?](http://www.gotfuturama.com/Information/Encyc-55-What_If_Machine/)" machine into XenServer to facilitate counter-factual reasoning.

When reconfiguring HA via XenCenter, you can supply a hypothetical series of VM priorities, and XenServer will return a number of host failures which would be tolerated under this scheme.  This lets you try various combinations of VM protections depending on your business needs, and see if the number of host failures is appropriate to the level of paranoia you desire.

This can even be done via the CLI, using the snappily named "**xe pool-ha-compute-max-host-failures-to-tolerate**" when HA is enabled.

The nice thing about XenServer HA is that it is done at the XenAPI level, and so  any of the standard clients (such as the xe CLI or XenCenter) or any third-party clients which use the XenAPI will all interoperate just fine.  The XenServer pool dynamically recalculates plans in response to the client requests, and so no special "oracle" is required outside of the pool to figure out HA plans.

Finally, HA makes master election completely invisible.  Any host in a pool can be a master host, and the pool database is constantly replicated across all nodes and also backed up to shared storage on the heartbeat SR for additional safety.  Any XenAPI client can connect to any host, and a redirect is issued to the current master host.

**Protection Levels**

Each VM in an HA pool can be either **fully protected**, **best-effort** or **unprotected**. VMs which are protected are all included in the failover planning, and if no plan exists for which they can all be reliably restarted then the pool is considered to be overcommitted.

Hugh Warrington (who implemented the XenCenter HA support) explained what use protection levels are:

*"Best-effort VMs are not considered when calculating a failover plan, but the pool will still try to start them as a one-off if a host that is running them fails.  This restart is attempted after all protected VMs are restarted, and if the attempt to start them fails then it will not be retried.  This is a useful setting for test/dev VMs which aren't critical to keep running, but would be nice to do so in a pool which also has some important VMs which absolutely must run."*

[](http://community.citrix.com/download/attachments/43614779/ha-wizard-5.png)

There are some advanced features which are only available via the CLI.   Each protected VM in an HA pool can be assigned a numeric ha-restart-priority.  If a pool is well-resourced with a high *nhtol*, then these restart priorities are not relevant: the VMs are all guaranteed to be started.

If more hosts fail than have been planned for, then the priorities are used to determine the order in which VMs are restarted.  This ensures that in over-committed pools, the most important VMs are restarted first.  Although the pool will start priority 1 VMs first, they might not finish booting before the priority 2 VMs, and so this should not be used as the basis for service ordering.

Note that it's very important to **ensure that a VM is agile** when protecting it by HA.  If the VM is not agile (e.g has a physical CD drive mapped in from a host), then it can only be assigned Best Effort restart since it is tied to one host!

**XenCenter support for HA**

The best practice for HA is not to make configuration changes while it is enabled.  Instead, it is intended to be the "2am safeguard" which will restart hosts in the event of a problem when there isn't a human administrator nearby.  If you are actively making configuration changes such as applying patches, then HA should be disabled for the duration of these changes.

XenCenter makes some common changes under HA much more user-friendly, which I asked [Ewan Mellor](http://community.citrix.com/blogs/citrite/ewanm/) (the principal GUI engineer) about:

* Normally a protected VM cannot be shut down via the CLI or from within the guest (a shutdown from within the guest will automatically restart it).  If you try to shutdown from XenCenter, it will give you the option of unprotecting the VM and then shutting it down first.  Thus, accidental in-guest shutdowns wont result in downtime, but administrators can still stop a protected guest if they really want to.
* If you want to reboot a host when HA is enabled, XenCenter automatically uses the hypothetical planning calculation to determine if this would invalidate the failover plan.  If it doesn't affect it, then the host is shut down normally.  If the plan would be violated, but the *nhtol* is greater than 1, XenCenter will give the administrator the option of lowering the *nhtol* value by 1.  This reduces the overall resilience of the pool, but always ensures that at least one host failure will be tolerated.  When the host comes back up, the plan is automatically recalculated and the original *nhtol* value restored if appropriate.
* If you try to apply a hotfix, then XenCenter will disable HA for the duration of the pool patching wizard.  It is important to manually keep an eye on hotfix application to ensure that host failures do not disrupt the operation of the pool.

So, I hope this short article has given you a taster... just kidding! This post is almost as long as my PhD thesis, but then, HA is a complex topic. Please do feel free to get back to me with comments and feedback about how we can improve it in the future releases, or if you just love it the way it is.  Many thanks to Dave Scott, Richard Sharp, Ewan Mellor and Hugh Warrington for their input to this article.